A Neural-Evolutionary Algorithm for Autonomous Transit Network Design

Andrew Holliday and Gregory Dudek[1]

Abstract— **Planning a public transit network is a challenging optimization problem, but essential in order to realize the benefits of autonomous buses. We propose a novel algorithm for planning networks of routes for autonomous buses. We first train a graph neural net model as a policy for constructing route networks, and then use the policy as one of several mutation operators in a evolutionary algorithm. We evaluate this algorithm on a standard set of benchmarks for transit network design, and find that it outperforms the learned policy alone by up to 20% and a plain evolutionary algorithm approach by up to 53% on realistic benchmark instances.**

I. INTRODUCTION

Mass public transit is key to sustainable urban design. Mass transit can reduce traffic congestion, thus shortening trips, and consume less energy, thus reducing emissions, versus even idealized point-to-point transit [1], [2], [3]. Self-driving vehicle technology is bringing with it autonomous bus fleets, which are already being deployed [4], [5], [6]. By dispensing with drivers, autonomous buses can enable fleets of more and smaller vehicles [7, chapter 9]. This can make service more frequent and reliable, two major factors in attracting riders to public transit [8]. But frequent service only helps those living near stops; and the need to transfer between lines makes transit less attractive [8], [9]. An efficient network also saves operating costs, which can then be spent on automating and expanding fleets. So in order to fully realize the benefits of autonomous buses, we need well-designed transit networks.

But the Transit Network Design Problem (NDP) is very challenging. It is similar to the travelling salesman problem (TSP) and vehicle routing problem (VRP), but is much more complex because of its many-to-many nature, and because of transfers between transit lines. Furthermore, real-world instances of this problem may have hundreds or thousands of transit stops [10]. The most successful approaches to date have been approximate solvers based on metaheuristics such as evolutionary algorithms (EAs). These work by repeatedly applying one or more low-level heuristics that randomly modify a solution, and guiding this random search towards more promising solutions over many iterations. We here wish to consider whether a neural net could learn to use information about the scenario to select promising changes - instead of purely random changes, as existing low-level heuristics do - and whether this could improve the performance of metaheuristic algorithms.

To this end, we train a graph neural net (GNN) policy via reinforcement learning (RL) to construct transit networks for

[1]Both authors are with the School of Computer Science, McGill University. They can be reached at `ahollid@cim.mcgill.ca` and `dudek@cim.mcgill.ca`, respectively.

cities. We then use this policy as a low-level heuristic in a EA. We compare our approach to the GNN model on its own and to the baseline evolutionary algorithm, and find that on realistically-sized problem instances, the hybrid algorithm performs best, and is competitive with other state-of-the-art approaches.

II. RELATED WORK

A. Graph Nets and Reinforcement Learning for Optimization Problems

Graph neural nets (GNNs) are neural net models that are designed to operate on graph-structured data [11], [12], [13], [14]. They were inspired by the success of convolutional neural nets on computer vision tasks and have been applied in many domains, including analyzing large web graphs [15], designing printed circuit boards [16], and predicting chemical properties of molecules [14], [17]. An overview of GNNs is provided by [18].

There is growing interest in the application of machine learning techniques to solve combinatorial optimization (CO) problems such as the TSP [19]. In CO problems generally, it is difficult to find a globally optimal solution but easy to compute a scalar quality metric for a given solution. As noted by [19], this makes RL, in which a system is trained to maximize a scalar reward, a good fit. Many CO problems have natural interpretations as graphs, making GNNs also a good fit. [20] proposed a GNN model called a Pointer Network, and trained it by supervised learning to solve TSP instances. Much recent work has built on this by using various RL algorithms to train GNN to construct CO solutions [21], [22], [23], [24]. These have attained impressive performance on the TSP, the VRP, and related problems. [25] use a hybrid of Monte Carlo Tree Search and Beam Search to guide the construction of Capacitated VRP (CVRP) solutions by a neural net policy. [26] trains an recurrent neural net (RNN) via RL to provide a starting population of solutions to a EA, the outputs of which are used to further train the RNN. [27] train a model on small TSP instances and propose an algorithm to apply it to much larger instances. These approaches belong to the family of "construction" methods, which solve a CO problem by starting with an "empty" solution and adding to it in steps, ending once the solution is complete - for example, when all nodes have been visited in the TSP. The solutions from these neural construction methods come close to the quality of those from specialized TSP algorithms such as Concorde [28], while requiring much less run-time to compute [22].

"Improvement" methods, by contrast with construction methods, start with a complete solution and repeatedly modify it, searching through the solution space for improvements.

Metaheuristics like EA belong to this category. These are more computationally costly than construction methods but can yield better solutions to CO problems. Some work has considered training neural nets to choose the search moves to be made at each step of an improvement method [29], [30], [31], [32], [33], and [34] train one GNN to construct a set of initial solutions, and another to modify and improve them. This work has shown impressive performance on classic CO problems like the TSP. Our approach belongs to this family, but is novel in that it uses an RL-trained GNN to perform mutations in a EA.

B. Optimization of Public Transit

The Transit Network Design Problem is NP-complete [35], making it impractical to solve optimally. While analytical optimization and mathematical programming methods have been successful on small instances [36], [37], they struggle to realistically represent the problem [38], [39], and so metaheuristic approaches (as defined by [40]) have been more widely applied. Historically, genetic algorithms (GAs), simulated annealing, and ant-colony optimization have been most popular, along with hybrids of these methods [38], [39], [41].

While neural nets have seen much use for predictive problems in urban mobility [42], [43], [44], [45], [46], [47] and for transit optimization problems such as scheduling and passenger flow control [48], [49], [50], [51], little work has applied RL or neural nets to the NDP. [52] and [53] both use RL to design a network and schedule for the Mandl benchmark [54], a single small graph with just 15 nodes. [52] use a GNN approach inspired by [22]; in our own work we experimented with a nearly identical approach to [52], but found it did not scale beyond very small instances. Meanwhile, [53] uses tabular RL, a type of approach which also scales poorly. Both these approaches require a new model to be trained on each problem instance. Our approach, by contrast, is able to find good solutions for realistically-sized NDP instances of more than 100 nodes, and can be applied to problem instances unseen during training.

In prior work [55], we trained a GNN construction method to plan a transit network, and used it to provide initial solutions to the EA of [41], finding that this improved the quality of the solutions found. The present work considers the same GNN architecture and training procedure, but uses the GNN to generate new routes throughout an improvement method, instead of just once at the outset.

III. THE TRANSIT NETWORK DESIGN PROBLEM

In the NDP, one is given an augmented graph that represents a city:

$$\mathcal{C} = (\mathcal{N}, \mathcal{E}_s, D) \tag{1}$$

This is comprised of a set $\mathcal{N}$ of n nodes, representing candidate stop locations; a set $\mathcal{E}_s$ of street edges (i, j, τ_{ij}) connecting the nodes, weighted by driving time τ_{ij}; and an $n \times n$ matrix D giving the travel demand (in number of trips) between every pair of nodes in $\mathcal{N}$. A route r is a sequence of nodes in $\mathcal{N}$. The goal is to find a set of routes $\mathcal{R}$ that
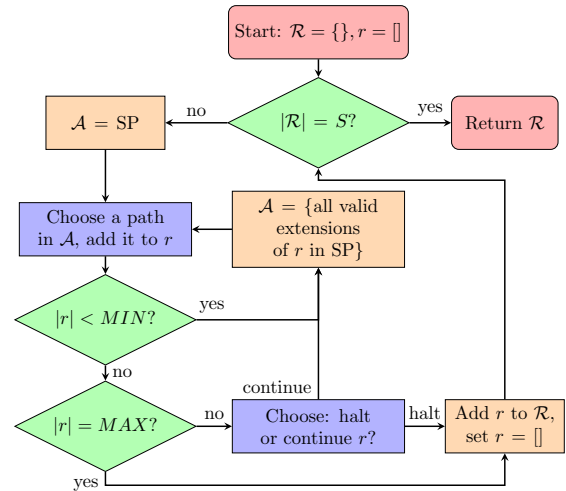

Fig. 1: A flowchart of the transit network construction process defined by our Markov Decision Process (MDP). Blue boxes indicate points where the timestep t is incremented and the neural net policy selects an action.

minimize a cost function $C : \mathcal{C}, \mathcal{R} \to \mathbb{R}^+$. The route set $\mathcal{R}$ is called a transit network, and is subject to the following constraints:

1) $\mathcal{R}$ must allow every node in $\mathcal{N}$ to be reached from every other node via transit.
2) $\mathcal{R}$ must contain exactly S routes (ie. $|\mathcal{R}| = S$), where S is a parameter set by the user.
3) $MIN \leq |r| \leq MAX \ \forall \ r \in \mathcal{R}$, where MIN and MAX are parameters set by the user.
4) A route $r \in \mathcal{R}$ may not contain cycles; each node $i \in \mathcal{N}$ can appear in r at most once.
5) A route cannot "skip" nodes: if nodes i, j are consecutive in $r \in \mathcal{R}$, then (i, j, τ_{ij}) must be in $\mathcal{E}_s$.

We here deal with the symmetric NDP, that is: $D = D^\top$, $(i, j, \tau_{ij}) \in \mathcal{E}_s$ iff. $(j, i, \tau_{ij}) \in \mathcal{E}_s$, and all routes are traversed both forwards and backwards by vehicles on them.

A. Markov Decision Process Formulation

A Markov Decision Process (MDP) is a formalism used to define problems in RL. In an MDP, an **agent** interacts with an environment over time steps t. At each t, the environment is in **state** $s_t \in \mathcal{S}$, and the agent takes some **action** $a_t \in \mathcal{A}_t$, where $\mathcal{A}_t$ is the set of possible actions at t. The environment transitions to a new state $s_{t+1} \in \mathcal{S}$ according to the state transition distribution $P(s'|s, a)$, and the agent receives a scalar **reward** $R_t \in \mathbb{R}$ according to the reward distribution $P(R|s, a, s')$. The agent chooses a_t according to its **policy** $\pi(a|s)$, a probability distribution over $\mathcal{A}_t$ given s_t. In RL, the goal is to learn a policy π that maximizes cumulative reward.

We here describe the MDP we use to represent a construction approach to the NDP, visualized in Fig. 1. The state s_t is composed of the set of routes $\mathcal{R}_t$ planned so far, and an

incomplete route r_t which is being planned.

$$s_t = (\mathcal{R}_t, r_t) \qquad (2)$$

The starting state is $s_0 = (\mathcal{R}_0 = \{\}, r_0 = [])$. The MDP alternates with t between two modes: on odd-numbered t, the agent chooses an extension to r_t; on even-numbered t, the agent chooses whether to finish r_t and start a new route.

On odd-numbered t, $\mathcal{A}_t$ is drawn from SP, the set of shortest paths between all node pairs. If $r_t = []$, then $\mathcal{A}_t = \{a \mid a \in \text{SP}, |a| \le MAX\}$. Otherwise, $\mathcal{A}_t$ is comprised of paths $a \in \text{SP}$ that meet the following conditions:

- $(i, j, \tau_{ij}) \in \mathcal{E}_s$, where i is the first node of a and j is the last node of r_t, or vice-versa
- a and r_t have no nodes in common
- $|a| \le MAX - |r_t|$

Once a path $a_t \in \mathcal{A}_t$ is chosen, r_{t+1} is formed by appending a_t to the beginning or end of r_t as appropriate.

On even-numbered t, the action space depends on the number of stops in r_t:

$$\mathcal{A}_t = \begin{cases} \{\text{continue}\} & \text{if} |r_t| < MIN \\ \{\text{halt}\} & \text{if} |r_t| = MAX \\ \{\text{continue}, \text{halt}\} & \text{otherwise} \end{cases} \qquad (3)$$

If $a_t = $ halt, r_t is added to $\mathcal{R}_t$ to get $\mathcal{R}_{t+1}$, and r_{t+1} is set to a new empty route. If $a_t = $ continue, then $\mathcal{R}_{t+1}$ and r_{t+1} are unchanged from step t.

When $|\mathcal{R}_t| = S$, the MDP terminates, giving the final reward $R_t = -C(\mathcal{C}, \mathcal{R}_t)$. At all prior steps, $R_t = 0$.

This MDP formulation imposes some helpful biases on $\mathcal{R}$. First, it biases routes towards directness by forcing them to be composed of shortest paths. Second, the alternation between deciding *whether* and *how* to extend a route means that the probability of halting does not depend on how many extensions are possible at t.

B. Cost Function

Following earlier work [56], the NDP cost function has three components. The passenger cost is the average transit trip time:

$$C_p(\mathcal{C}, \mathcal{R}) = \frac{\sum_{i,j} D_{ij} \tau_{\mathcal{R}ij}}{\sum_{i,j} D_{ij}} \qquad (4)$$

Where $\tau_{\mathcal{R}ij}$ is the time of the shortest transit trip from i to j over $\mathcal{R}$, including a time penalty p_T for each transfer.

The operator cost is the total driving time of the routes:

$$C_o(\mathcal{C}, \mathcal{R}) = \sum_{r \in \mathcal{R}} \tau_r \qquad (5)$$

Where τ_r is the time needed to completely traverse a route r in both directions.

To enforce the constraints on $\mathcal{R}$, we add a third term C_c, which is the fraction of node pairs that are not connected by $\mathcal{R}$ plus a measure of how much $|r| > MAX$ or $|r| < MIN$ across all routes. The cost function is then:

$$C(\mathcal{C}, \mathcal{R}) = \alpha w_p C_p + (1 - \alpha) w_o C_o + \beta C_c \qquad (6)$$

The weight $\alpha \in [0, 1]$ controls the trade-off between passenger and operator costs. w_p and w_o are re-scaling constants chosen so that $w_p C_p$ and $w_o C_o$ both vary roughly over the range $[0, 1]$ for different $\mathcal{C}$ and $\mathcal{R}$; this is done so that α will properly balance the two, and to stabilize training of the GNN policy. The values used are $w_p = (\max_{i,j} T_{ij})^{-1}$ and $w_o = (3S \max_{i,j} T_{ij})^{-1}$, where T is an $n \times n$ matrix of shortest-path driving times between every node pair.

IV. METHODOLOGY

A. Learned Constructor

We propose to learn a policy $\pi_\theta(a|s)$ with the objective of maximizing the cumulative return G on the construction MDP described in section III-A. By then evaluating this policy on the MDP for some city $\mathcal{C}$, we can obtain a transit network $\mathcal{R}$ for that city. We denote this algorithm the Learned Constructor (LC).

The policy π_θ is a neural net parameterized by θ. Its "backbone" is a graph attention net [57] which treats the city as a fully-connected graph on the nodes $\mathcal{N}$, where each edge has a feature vector e_{ij} containing information about demand, existing transit connections, and the street edge (if one exists) between i and j. We note that a graph attention net operating on a fully-connected graph has close parallels to a Transformer model [58], but unlike a Transformer it can use make use of edge features that describe relationships between nodes.

The backbone GNN outputs node embeddings Y, which are operated on by one of two policy "heads", depending on the timestep: NN_{ext} for choosing among extensions when t is odd, and NN_{halt} for deciding whether to halt when t is even. For the full details of the neural architecture, we direct the reader to our code release[2].

1) Training: Following the work of [22], we train the policy net using the policy gradient method REINFORCE with baseline [59], setting $\gamma = 1$. Since the reward R_t for the last step is the negative cost and at all other steps $R_t = 0$, this implies the return G_t at each timestep is simply:

$$G_t = \sum_{t'} \gamma^{t'-t} R_t = \sum_{t'} R_t = -C(\mathcal{C}, \mathcal{R}) \qquad (7)$$

The learning signal for each action a_t is $G_t - b(\mathcal{C}, \alpha)$, where the baseline $b(\mathcal{C}, \alpha)$ is a separate Multi-Layer Perceptron trained to predict the final reward obtained by the current policy for a given cost weight α and city $\mathcal{C}$.

We train the model on a dataset of synthetic cities. For each batch, a full rollout of the MDP is performed on the cities in the batch, $C(\mathcal{C}, \mathcal{R})$ is computed across the batch, and back-propagation and weight updates are applied to both the policy net and the baseline net.

To construct a synthetic city for the training dataset, we first generate its nodes and street network using one of these processes chosen at random:

- 4-nn: Sample n random points in a square to give $\mathcal{N}$. Add edges to each node i from its 4 nearest neighbours.

---

[2]Available at `https://www.cim.mcgill.ca/~mrl/hgrepo/transit_learning/`

- **4-grid:** Place n nodes in a rectangular grid as close to square as possible. Add edges from each node to its horizontal and vertical neighbours.
- **8-grid:** Like 4-grid, but also add edges between diagonal neighbours.
- **Voronoi:** Sample m random 2D points, and compute their Voronoi diagram [60]. Take the shared vertices and edges of the resulting Voronoi cells as $\mathcal{N}$ and $\mathcal{E}_s$. m is chosen so $|\mathcal{N}| = n$.

For each process except Voronoi, each edge in $\mathcal{E}_s$ is then deleted with user-defined probability ρ. If the resulting street graph is not connected, it is discarded and the process is repeated. Nodes are sampled in a 30km $\times$ 30km square, and a fixed vehicle speed of $v = 15$m/s is assumed to compute street edge weights $\tau_{ij} = ||(x_i, y_i) - (x_j, y_j)||_2/v$. Finally, we generate D by setting diagonal demands $D_{ii} = 0$ and uniformly sampling off-diagonal elements D_{ij} in the range $[60, 800]$.

All neural net inputs are normalized so as to have unit variance and zero mean across the entire dataset during training. The normalization parameters are saved as part of the model and applied to new data presented at test time.

B. Evolutionary Algorithm

An evolutionary algorithm is an improvement method in which an initial population of B solutions go through repeated stages of modification and selection based on their "fitness", thus increasing the fitness of the population over many iterations. In [41], an EA is presented for the NDP, which consists of alternating mutation and selection stages. In the mutation stage, two different mutation operators - type 1 and 2 - are each applied N_m times to half of the population. Each mutation is kept if it decreases the cost C_b of solution b. In the selection stage, each solution b "dies" with probability increasing with C_b. Dead solutions are replaced with copies ("offspring") of surviving solutions to maintain the population size B, with probability inversely proportional to the survivors' C_b. These alternating stages repeat for a fixed number of iterations I.

Each mutator begins by selecting a random terminal i on a random route r in the solution. The type-1 mutator then selects a random node $j \neq i$ in $\mathcal{N}$, and replaces r with the shortest path between i and j. The type-2 mutator chooses with probability 0.2 to delete i from r; otherwise, it adds a random node j in i's neighbourhood to r (before i if i is the first node in r, and after i if i is the last node in r), making j the new terminal.

We modify this algorithm by replacing the type-1 mutator with a "neural mutator". This mutator selects a random route r from $\mathcal{R}$, and then rolls out our learned policy π_θ starting from $\mathcal{R} \setminus r$, generating one new route r' that replaces r. The algorithm is otherwise unchanged. We replace the type-1 mutator because its action space (replacing one route by a shortest path) is a subset of the action space of the neural mutator (replacing one route by a new route composed of shortest paths), while the type-2 mutator's action space is quite

TABLE I: Statistics of the five synthetic benchmark cities used in our experiments.

City	n	$|\mathcal{E}_s|$	S	MIN	MAX	Area (km²)
Mandl	15	20	6	2	8	352.7
Mumford0	30	90	12	2	15	354.2
Mumford1	70	210	15	10	30	858.5
Mumford2	110	385	56	10	22	1394.3
Mumford3	127	425	60	12	25	1703.2

different. In this way, π_θ, which was trained as a construction policy, is used in an improvement method.

We note that in [41], the algorithm is described as a "bee colony optimization" algorithm. But "bee colony optimization" is merely a relabelling of the evolutionary-algorithm metaheuristic - the two are functionally identical [61]. To avoid the profusion of unnecessary terminology, we here describe [41]'s method as an EA.

V. EXPERIMENTS

All evaluations are performed on the Mandl [54] and Mumford [62] city datasets, two popular benchmarks for evaluating NDP algorithms [56], [63], [64], [65]. The Mandl dataset is one small synthetic city, while the Mumford dataset consists of four synthetic cities, labelled Mumford0 through Mumford3, and it gives values of S, MIN, and MAX to use when benchmarking on each city. The values n, S, MIN, and MAX for Mumford1, Mumford2, and Mumford3 are taken from three different real-world cities and their existing transit networks, giving the dataset a degree of realism. Details of these benchmarks are given in Table I.

In all experiments, we use policies π_θ that we trained on a dataset of 2^{15} synthetic cities with $n = 20$. A 90:10 training:validation split of this dataset is used; after each epoch of training, the model is evaluated on the validation set, and at the end of training, the parameters θ from the epoch with the best validation-set performance are returned. Data augmentation is applied each time a city is used for training. This consists of multiplying the node positions (x_i, y_i) and travel times τ_{ij} by a random factor $c_s \sim [0.4, 1.6]$, rotating the node positions about their centroid by a random angle $\phi \sim [0°, 360°)$, and multiplying D by a random factor $c_d \sim [0.8, 1.2]$. During training, we use constant values $S = 10, MIN = 2, MAX = 15$; we used these as initial values based on the values for the similarly-sized Mandl and Mumford0 cities, and found they gave good results, so did not vary them. We sample a different $\alpha \sim [0, 1]$ for each city in a batch, so that the policy will learn to condition its actions on the user preferences implicit in the cost function. Training proceeds for 5 epochs, with a batch size of 64 cities. When performing multiple training runs with different random seeds, we hold the dataset constant, but not the data augmentation.

We henceforth refer to [41]'s evolutionary algorithm as EA, and to our own algorithm as NEA. For both, we set all algorithmic parameters to the values used in the experiments of [41]: population size $B = 10$, mutations per mutation stage per individual $N_m = 10$, and total number of mutate-select

iterations $I = 400$. We tried running EA for up to $I = 2,000$ on several cities, but found this did not yield any improvement over $I = 400$. In each mutation stage, the neural mutator is applied to half of the population, and the type-2 mutator to the other half, just as the base EA splits type-1 and type-2 mutators equally among individuals. Hyperparameter settings of the model architecture of π_θ and training process were arrived at by a limited manual search; for their values, we direct the reader to the configuration files contained in our code release. We set the constraint penalty weight $\beta = 5$ in all experiments.

A. Results

We compare LC, EA, and NEA on Mandl and the four Mumford cities. To evaluate LC, we perform 100 rollouts on each city and choose the $\mathcal{R}$ from among them that minimizes $C(\mathcal{C}, \mathcal{R})$ (denoted LC-100). Each algorithm is run across a range of 10 random seeds, using separate policy parameters θ trained with that seed. We report results averaged over all of the seeds. Our main results are summarized in Table II, which shows results at three different α values, optimizing for the operators' perspective ($\alpha = 0.0$), the passengers' perspective ($\alpha = 1.0$), and a balance of the two ($\alpha = 0.5$).

The results show that while EA performs best on the two smallest cities in most cases, its relative performance worsens considerably when $n \geq 70$. On Mumford1, 2, and 3, for each α, NEA performs best overall on these three cities. It is better than LC-100 in every instance, improving on its cost by about 6% $\alpha = 1.0$ and 0.5, and by up to 20% at $\alpha = 0.0$; and it improves on EA by 33% to 53% on Mumford3 depending on α.

With $\alpha = 0.0$, NEA failed to obey route length limits with 1 of the 10 seeds. This may be because at $\alpha = 0.0$, the reduction in C_o given by under-length routes overwhelms the cost penalty due to a few routes being too long. This might be resolved by simply increasing β or adjusting the specific form of C_c.

We also observed that while one NEA iteration is somewhat slower than one EA iteration due to the need to execute π_θ, NEA's cost falls much more rapidly with each iteration than does EA's, so much so that NEA can achieve lower-cost solutions in considerably less time than EA. This fact may be practically significant: autonomous buses may enable new forms of transit involving online re-planning of bus routes, and in this application, planning high-quality route systems at speed will be key. NEA may make such an application feasible where EA may not.

B. Trade-offs Between Passenger and Operator Costs

There is always a trade-off between minimizing the passenger cost C_p and the operator cost C_o: making transit routes longer increases C_o, but allows more and faster direct connections between stops, and so may decrease C_p. The weight α can be set by the user to indicate how much they care about C_p versus C_o, and each algorithm's behaviour will change accordingly. Fig. 2 illustrates the trade-offs made by the different methods, as we vary α over the range $[0, 1]$

in steps of 0.1 on the three cities with real-world statistics, Mumford1, 2, and 3. NEA's solutions not only dominate those of EA, but also achieve a much wider range of C_p and C_o than either of EA or LC-100, which will be more satisfactory if the user cares only about one or the other component.

Both LC and EA have more narrow ranges of C_p and C_o on these three cities, but the ranges are mostly non-overlapping. Some of NEA's greater range seems to be due to combining the non-overlapping ranges of the constituent parts, but NEA's range is greater than the union of LC's and EA's ranges. This implies that the larger action space of the neural mutator versus the type-1 mutator allows NEA to explore a much wider range of solutions by taking wider "steps" in solution space.

C. Comparison with Other Methods

The main purpose of our work was to see whether a neural net could learn route-building heuristics that would improve the results of an existing metaheuristic algorithm, and we have shown that it does. A direct comparison with other non-neural algorithms from the literature is secondary to this purpose, but is still of some interest. So Table III compares our NEA with more methods from elsewhere in the literature, on the three real-world-based Mumford benchmark cities. We bold the best result, and show NEA's result in blue if it is within one standard deviation of the best result. In [56], [63], results are reported for $\alpha = 0.0$ (optimizing only C_o) and $\alpha = 1.0$ (optimizing only C_p), while [64] only reports results for $\alpha = 1.0$.

At $\alpha = 0.0$, we find that NEA performs comparably with [56], but not as well as [63]. At $\alpha = 1.0$, NEA fares better, outperforming [56], [63], and [64] on two of the three cities. We note that this competitive performance is impressive given the substantial difference in runtime. While [56] do not report run-times with their results, [63] report that their NSGA-II genetic algorithm takes more than two days to run on Mumford3, while [64] report that their procedure takes eight hours just to construct the initial solution for Mumford3. By comparison, our NEA runs take less than 20 minutes on a desktop computer with a 2.4 GHz Intel i9-12900F processor and an NVIDIA RTX 3090 GPU. Running our method with larger values of I or B would increase runtime but may generate better solutions. Using other, more costly but more powerful metaheuristics with our neural heuristic may also give better results. We leave this to future work.

VI. Discussion

We have shown that a neural net policy trained to construct whole solutions to the transit network design problem learns broadly useful heuristics for planning transit routes. This allows it serve as a mutation operator in an evolutionary algorithm, significantly improving the quality of the discovered solutions on a standard benchmark. Meanwhile, the evolutionary algorithm's power as a search method enhances results considerably over those achieved by directly sampling solutions from the learned policy.

TABLE II: Average final cost $C(\mathcal{C}, \mathcal{R})$ achieved by each method over 10 random seeds, for three different settings of cost weight α. Bold indicates the best in each column. Orange indicates that one seed's solution violated a constraint, red indicates two or three seeds' solutions did so. Values after $\pm$ are standard deviations over the 10 seeds.

α	Method	City:	Mandl	Mumford0	Mumford1	Mumford2	Mumford3
	EA		$\mathbf{0.270 \pm 0.048}$	$\mathbf{0.272 \pm 0.044}$	0.854 ± 0.277	0.692 ± 0.277	0.853 ± 0.297
0.0	LC-100		0.317 ± 0.076	0.487 ± 0.323	0.853 ± 0.371	0.688 ± 0.180	0.710 ± 0.172
	NEA		0.276 ± 0.054	0.298 ± 0.122	$\mathbf{0.623 \pm 0.163}$	$\mathbf{0.537 \pm 0.236}$	$\mathbf{0.572 \pm 0.262}$
	EA		$\mathbf{0.327 \pm 0.009}$	$\mathbf{0.563 \pm 0.009}$	1.015 ± 0.424	0.710 ± 0.258	0.944 ± 0.274
0.5	LC-100		0.346 ± 0.040	0.638 ± 0.138	0.742 ± 0.176	0.617 ± 0.085	0.612 ± 0.082
	NEA		0.331 ± 0.021	0.571 ± 0.034	$\mathbf{0.627 \pm 0.049}$	$\mathbf{0.532 \pm 0.028}$	$\mathbf{0.584 \pm 0.215}$
	EA		$\mathbf{0.315 \pm 0.002}$	0.645 ± 0.030	0.739 ± 0.271	0.656 ± 0.252	1.004 ± 0.401
1.0	LC-100		0.340 ± 0.015	0.738 ± 0.042	0.600 ± 0.017	0.534 ± 0.009	0.504 ± 0.007
	NEA		0.317 ± 0.004	$\mathbf{0.637 \pm 0.036}$	$\mathbf{0.564 \pm 0.018}$	$\mathbf{0.507 \pm 0.007}$	$\mathbf{0.481 \pm 0.007}$

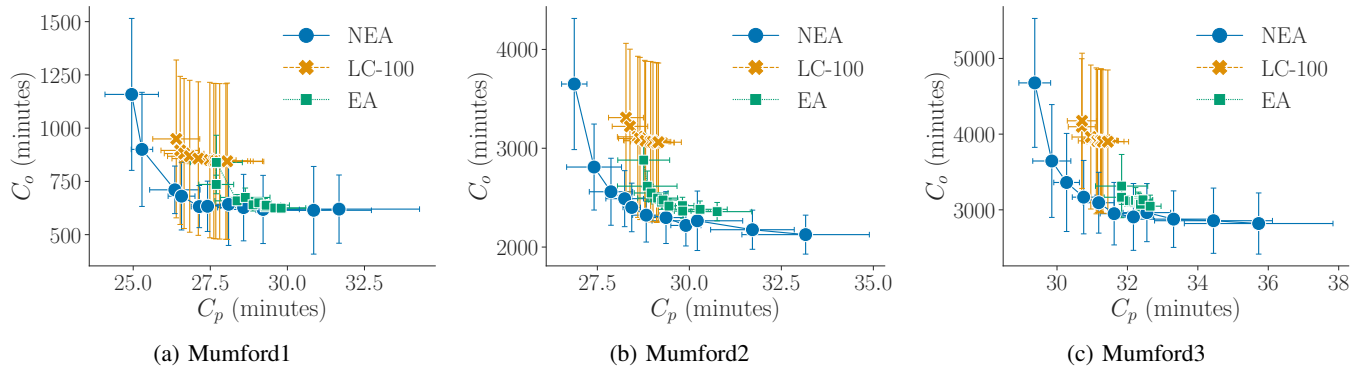(a) Mumford1   (b) Mumford2   (c) Mumford3

Fig. 2: Trade-offs achieved by different methods between passenger cost C_p (on the x-axis) and operator cost C_o (on the y-axis), across values of α evenly spaced over the range $[0, 1]$, averaged over 10 random seeds. Both axes have units of minutes. Error bars show one standard deviation over the ten random seeds for each point. We wish to minimize both values, so the lower-left direction in each plot represents improvement. A line links two points if they have adjacent α values, so these curves show a smooth progression from low C_o (at the right) to low C_p (at the left) as α increases.

TABLE III: Comparison between our method and others from the literature on the three Mumford cities, for $\alpha = 0.0$ and $\alpha = 1.0$, with values for the corresponding metric.

Method	Mumford1	Mumford2	Mumford3
	C_o at $\alpha = 0.0$		
[56]	568	2244	2830
[63]	**462**	**1875**	**2301**
NEA	620 ± 160	2126 ± 197	2819 ± 404
	C_p at $\alpha = 1.0$		
[56]	24.79	28.65	31.44
[63]	23.91	27.02	29.50
[64]	**23.25**	**26.82**	30.41
NEA	24.95 ± 0.87	26.87 ± 0.34	$\mathbf{29.37 \pm 0.45}$

Better results could likely be achieved by training a policy directly in the context of an improvement process, rather than training it in a construction process as was done here. It would also be interesting to use an ensemble of separately-trained policies as different heuristics within a GA, as opposed to the single model used in our experiments. Training over variable n, S, MIN, and MAX would be interesting as well.

We note that both the construction policy and the neural

evolutionary algorithm outperform the plain evolutionary algorithm on the Mumford1, 2, and 3 problem instances - all those that were modeled on specific real-world cities in scale. Furthermore, the gap between these grows with the size of the city. This suggests that our approach may scale better to much larger problem sizes. This is significant, as many real-world cities have hundreds or thousands of bus stop locations [10]. In future, we intend to evaluate our technique on data from real-world cities.

Another promising direction would be to extend this work to the problem of planning a transit network that would operate in conjunction with a mobility-on-demand system composed of autonomous taxis. A fully-autonomous urban transit system will need to make use of both point-to-point autonomous ride-sharing and fixed autonomous mass transit routes to be as useful and efficient as possible [66], [67]. The system proposed in this paper could serve as part of an algorithm for planning such a combined autonomous transit system. Future work could explore how our method could be combined with techniques from the literature on autonomous mobility-on-demand planning [68], [69], [70].

REFERENCES

[1] T. Roughgarden and É. Tardos, "How bad is selfish routing?" *Journal of the ACM (JACM)*, vol. 49, no. 2, pp. 236–259, 2002.

[2] S. Oh, R. Seshadri, D.-T. Le, P. C. Zegras, and M. E. Ben-Akiva, "Evaluating automated demand responsive transit using microsimulation," *IEEE Access*, vol. 8, pp. 82 551–82 561, 2020.

[3] J. Rich, R. Seshadri, A. J. Jomeh, and S. R. Clausen, "Fixed routing or demand-responsive? agent-based modelling of autonomous first and last mile services in light-rail systems," *Transportation Research Part A: Policy and Practice*, vol. 173, p. 103676, 2023.

[4] A. France-Presse, "Driverless electric bus hits the road in Spanish city of Málaga," https://www.theguardian.com/world/2021/feb/25/driverless-electric-bus-hits-the-road-in-spanish-city-of-malaga, 2 2021, accessed: 2021-03-09.

[5] J. Plautz, "Autonomous shuttles launch in Detroit," https://www.smartcitiesdive.com/news/autonomous-shuttles-launch-in-detroit/526999/, 7 2018, accessed: 2019-11-27.

[6] R. Beene, "A Florida monorail makes way for the robot bus of tomorrow," *Bloomberg*, 2 2018.

[7] A. Ceder, *Public Transit Planning and Operation: Modeling, Practice and Behaviour*. CRC Press, 2016.

[8] S. Chakrabarti, "How can public transit get people out of their cars? An analysis of transit mode choice for commute trips in los angeles," *Transport Policy*, vol. 54, no. October 2016, pp. 80–89, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.tranpol.2016.11.005

[9] A. El-Geneidy, M. Grimsrud, R. Wasfi, P. Tétreault, and J. Surprenant-Legault, "New evidence on walking distances to transit stops: Identifying redundancies and gaps using variable service areas," *Transportation*, vol. 41, no. 1, pp. 193–210, 2014.

[10] Société de transport de Montréal, "Everything about the stm," 2013, accessed: 2023-05-17. [Online]. Available: https://web.archive.org/web/20130610123159/http://www.stm.info/english/en-bref/a-toutsurlaSTM.htm

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *CoRR*, vol. abs/1606.09375, 2016. [Online]. Available: http://arxiv.org/abs/1606.09375

[14] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in neural information processing systems*, vol. 28, 2015.

[15] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," *CoRR*, vol. abs/1806.01973, 2018. [Online]. Available: http://arxiv.org/abs/1806.01973

[16] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1263–1272. [Online]. Available: https://proceedings.mlr.press/v70/gilmer17a.html

[18] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[19] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[20] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *arXiv preprint arXiv:1506.03134*, 2015.

[21] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *arXiv preprint arXiv:1704.01665*, 2017.

[22] W. Kool, H. V. Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *ICLR*, 2019.

[23] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *International Conference on Learning Representations*, 2019.

[24] Q. Sykora, M. Ren, and R. Urtasun, "Multi-agent routing value iteration network," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9300–9310.

[25] J. Choo, Y.-D. Kwon, J. Kim, J. Jae, A. Hottung, K. Tierney, and Y. Gwon, "Simulation-guided beam search for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8760–8772, 2022.

[26] T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen, "Symbolic regression via neural-guided genetic programming population seeding," *arXiv preprint arXiv:2111.00053*, 2021.

[27] Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large tsp instances," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 8, 2021, pp. 7474–7482.

[28] D. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, "Concorde tsp solver," 2001. [Online]. Available: https://www.math.uwaterloo.ca/tsp/concorde/index.html

[29] A. Hottung and K. Tierney, "Neural large neighborhood search for the capacitated vehicle routing problem," *arXiv preprint arXiv:1911.09539*, 2019.

[30] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] P. R. d O Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Asian conference on machine learning*. PMLR, 2020, pp. 465–480.

[32] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 9, pp. 5057–5069, 2021.

[33] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 096–11 107, 2021.

[34] M. Kim, J. Park, *et al.*, "Learning collaborative policies to solve np-hard routing problems," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 418–10 430, 2021.

[35] C. Quak, "Bus line planning," *A passenger-oriented approach of the construction of a global line network and an efficient timetable. Master's thesis, Delft University, Delft, Netherlands*, 2003.

[36] R. van Nes, "Multiuser-class urban transit network design," *Transportation Research Record*, vol. 1835, no. 1, pp. 25–33, 2003. [Online]. Available: https://doi.org/10.3141/1835-04

[37] J. Guan, H. Yang, and S. Wirasinghe, "Simultaneous optimization of transit line configuration and passenger line assignment," *Transportation Research Part B: Methodological*, vol. 40, pp. 885–902, 12 2006.

[38] V. Guihaire and J.-K. Hao, "Transit network design and scheduling: A global review," *Transportation Research Part A: Policy and Practice*, vol. 42, no. 10, pp. 1251–1273, 2008.

[39] K. Kepaptsoglou and M. Karlaftis, "Transit route network design problem: Review," *Journal of Transportation Engineering*, vol. 135, no. 8, pp. 491–505, 2009.

[40] K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," in *Handbook of heuristics*. Springer, 2018, pp. 791–808.

[41] M. Nikolić and D. Teodorović, "Transit network design by bee colony optimization," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5945–5955, 2013.

[42] Y. Xiong and J. B. Schneider, "Transportation network design using a cumulative genetic algorithm and neural network," *Transportation Research Record*, vol. 1364, 1992.

[43] J.-P. Rodrigue, "Parallel modelling and neural networks: An overview for transportation/land use systems," *Transportation Research Part C: Emerging Technologies*, vol. 5, no. 5, pp. 259–271, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X97000144

[44] S. I.-J. Chien, Y. Ding, and C. Wei, "Dynamic bus arrival time prediction with artificial neural networks," *Journal of transportation engineering*, vol. 128, no. 5, pp. 429–438, 2002.

[45] R. Jeong and R. Rilett, "Bus arrival time prediction using artificial neural network model," in *Proceedings. The 7th international IEEE conference on intelligent transportation systems (IEEE Cat. No. 04TH8749)*. IEEE, 2004, pp. 988–993.

[46] M. Y. Çodur and A. Tortum, "An artificial intelligent approach to traffic accident estimation: Model development and application," *Transport*, vol. 24, no. 2, pp. 135–142, 2009.

[47] C. Li, L. Bai, W. Liu, L. Yao, and S. T. Waller, "Graph neural network for robust public transit demand prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[48] L. Zou, J.-m. Xu, and L.-x. Zhu, "Light rail intelligent dispatching system based on reinforcement learning," in *2006 International Conference on Machine Learning and Cybernetics*, 2006, pp. 2493–2496.

[49] G. Ai, X. Zuo, G. Chen, and B. Wu, "Deep reinforcement learning based dynamic optimization of bus timetable," *Applied Soft Computing*, vol. 131, p. 109752, 2022.

[50] H. Yan, Z. Cui, X. Chen, and X. Ma, "Distributed multiagent deep reinforcement learning for multiline dynamic bus timetable optimization," *IEEE Transactions on Industrial Informatics*, vol. 19, pp. 469–479, 2023.

[51] Z. Jiang, W. Fan, W. Liu, B. Zhu, and J. Gu, "Reinforcement learning approach for coordinated passenger inflow control of urban rail transit in peak hours," *Transportation Research Part C: Emerging Technologies*, vol. 88, pp. 1–16, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X18300111

[52] A. Darwish, M. Khalil, and K. Badawi, "Optimising public bus transit networks using deep reinforcement learning," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[53] S. Yoo, J. B. Lee, and H. Han, "A reinforcement learning approach for bus network design and frequency setting optimisation," *Public Transport*, pp. 1–32, 2023.

[54] C. E. Mandl, "Evaluation and optimization of urban public transportation networks," *European Journal of Operational Research*, vol. 5, no. 6, pp. 396–404, 1980.

[55] A. Holliday and G. Dudek, "Augmenting transit network design algorithms with deep learning," in *2023 26th IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023.

[56] C. L. Mumford, "New heuristic and evolutionary operators for the multi-objective urban transit routing problem," in *2013 IEEE congress on evolutionary computation*. IEEE, 2013, pp. 939–946.

[57] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2021. [Online]. Available: https://arxiv.org/abs/2105.14491

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[59] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[60] S. Fortune, "Voronoi diagrams and delaunay triangulations," *Computing in Euclidean geometry*, pp. 225–265, 1995.

[61] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[62] C. L. Mumford, "Supplementary material for: New heuristic and evolutionary operators for the multi-objective urban transit routing problem, cec 2013," https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/CEC2013Supp.zip, 2013, accessed: 2023-03-24.

[63] M. P. John, C. L. Mumford, and R. Lewis, "An improved multi-objective algorithm for the urban transit routing problem," in *Evolutionary Computation in Combinatorial Optimisation*, C. Blum and G. Ochoa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 49–60.

[64] F. Kılıç and M. Gök, "A demand based route generation algorithm for public transit network design," *Computers & Operations Research*, vol. 51, pp. 21–29, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054814001300

[65] H. Lin and C. Tang, "Analysis and optimization of urban public transport lines based on multiobjective adaptive particle swarm optimization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 16 786–16 798, 2022.

[66] M. J. Alonso-González, T. Liu, O. Cats, N. Van Oort, and S. Hoogendoorn, "The potential of demand-responsive transport as a complement to public transport: An assessment framework and an empirical evaluation," *Transportation Research Record*, vol. 2672, no. 8, pp. 879–889, 2018.

[67] G. Leich and J. Bischoff, "Should autonomous shared taxis replace buses? a simulation study," *Transportation Research Procedia*, vol. 41, pp. 450–460, 2019.

[68] A. Vakayil, W. Gruel, and S. Samaranayake, "Integrating shared-vehicle mobility-on-demand systems with public transit," Tech. Rep., 2017.

[69] C. Ruch, S. Hörl, and E. Frazzoli, "Amodeus, a simulation-based testbed for autonomous mobility-on-demand systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3639–3644.

[70] P. Martí, J. Jordán, F. De la Prieta, H. Billhardt, and V. Julian, "Demand-responsive shared transportation: a self-interested proposal," *Electronics*, vol. 11, no. 1, p. 78, 2021.